# СИСТЕМА РАСПРЕДЕЛЕННЫХ ВЫЧИСЛЕНИЙ НА БАЗЕ ГЕНЕТИЧЕСКОГО АЛГОРИТМА

**В. Ю. Сакович, М. В. Яблонская**

*Белорусский государственный университет*
*Минск, Беларусь*
*e-mail: sakovich@bsu.by*

Описана система распределенных вычислений, позволяющая автоматизировать процесс проведения операций различных реализаций генетического алгоритма. Приведен анализ построеной системы и дана оценка качества ее работы.

*Ключевые слова*: распределенная система; распределенные вычисления; генетические алгоритмы.

# DISTRIBUTED COMPUTING SYSTEM BASED ON GENETIC ALGORITH

**V. Y. Sakovich, M. V. Yablonskaya**

*Belarusian State University*
*Minsk, Belarus*

The article describes a distributed computing system, which helps to automate operations performing of various realizations of the genetic algorithm. Given an analysis of constructed system and evaluation of its work quality.

*Keywords*: distributed system; distributed computing; genetic algorithms.

## ВВЕДЕНИЕ

Неуклонный рост необходимости проведения большого объема вычислений и решения трудоемких задач, который наблюдается в последние десятилетия, сделал распределенные вычисления популярными и востребованными. Возможность неограниченно наращивать производительность за счет добавления новых вычислительных единиц в распределенную систему позволяет достичь высоких мощностей, нужных для решения множества задач.

Одной из наиболее востребованных групп задач являются задачи оптимизации. При этом зачастую приходится искать решение таких задач на большом пространстве поиска и при большом количестве параметров. В этом случае широко используются генетические алгоритмы (ГА). Как правило, для нетривиальных проблем выполнение ГА требует значительных вычислительных ресурсов. Одним из способов улучшения скорости вычислений ГА является использование распределенных вычислений. Ниже будет описана созданная распределенная система, дающая возможность эффективно проводить операции разнообразных реализаций ГА.

## РЕАЛИЗОВАННОЕ ПРИЛОЖЕНИЕ

Реализованная распределенная система позволяет пользователю определять структуру особей для некоторой задачи, которая будет решаться с помощью генетического алгоритма. На основе описанной особи система производит итерационные действия алгоритма, так называемый эволюционный процесс, скрывая от пользователя реализацию этих действий. При этом для пользователя имеется возможность задавать параметры алгоритма, такие как точность, размер популяции, способ отбора и т. п. В сеть системы может быть включено любое количество машин, при этом есть возможность добавлять машины во время выполнения действий ГА.

Приложение реализовано на языке программирования Java. Его выбор был обусловлен наличием технологии динамической загрузки файлов и рефлексии, а также простотой создания и управления сетевыми соединениями на базе сокетов.

Для работы с системой ее необходимо предварительно сконфигурировать. Между компьютерами должна быть настроена сеть с возможностью взаимодействия по стандартам TCP/IP. На каждом из компьютеров будет запускаться по экземпляру приложения, поэтому для них необходимо прописать информацию о портах и ip-адресах остальных узлов и пометить главный узел в специальном файле адресов узлов. Такой файл должен быть описан для каждого экземпляра приложения. После этого на компьютерах запускаются процессы, которые выполняют подключение к главному узлу. Далее пользователь создает Java-класс, описывающий особь, который является наследником класса *AbstractIndivid. AbstractIndivid* – класс, созданный специально для описываемого приложения. В нем имеются абстрактные методы, представляющие собой действия над индивидом в ГА, т. е. случайная генерация, мутация, скрещивание с другим индивидом, подсчет целевой функции и т. д. За счет определения этих методов пользователь даст системе полную информацию о том, как оперировать созданным им определением особи.

На главном компьютере отображается интерфейс приложения. Пользователь передает jar-file, содержащий класс особи и вспомогательные классы (они должны быть сериализуемые), выбирает способ распределения и задает параметры алгоритма**.** При завершении ввода всех параметров запускается работа генетического алгоритма. Пользователю остается подождать, пока работа ГА не будет завершена и на главном узле не будет выведен результат работы, т. е. наилучшая из сгенерированных особей.

В системе реализованы несколько схем, по которым распределяются данные и взаимодействуют узлы системы.

## СХЕМЫ ВЗАИМОДЕЙСТВИЯ УЗЛОВ СИСТЕМЫ

*Схема «Мастер – рабочие».* В реальных задачах особи зачастую имеют сложную структуру и зависят от множества параметров, поэтому одной из самых трудоемких операций ГА является вычисление значений функции приспособленности. С целью ускорения процесса подсчета целесообразно распределить особи между несколькими машинами. Для этого используется система, где на одном узле (назовем его мастером) производятся операции скрещивания, мутации и отбора для всей популяции, а на остальных (назовем их рабочими) подсчет значений целевой функции части популяции. На каждом шаге эволюционного процесса на мастере сначала происходит деление популяции на части и каждому из рабочих посылается своя часть популяции. Деление происходит пропорционально памяти рабочих. Каждый рабочий считает значения

функции приспособленности и посылает свою часть популяции обратно мастеру. Далее мастер проводит операции скрещивания, мутации и отбора. Описанный процесс повторяется до тех пор, пока не будет получено решение, удовлетворяющее указанной пользователем точности.

В случае отказа какого-либо рабочего его часть популяции перераспределяется между остальными рабочими. Если выходит из строя мастер, то первый в списке адресов узлов рабочий берет его роль на себя. Остальные рабочие подключаются к нему и пересылают ему сохраненную во время последнего подсчета целевых функций часть популяции. В случае неудачи рабочие подключаются к следующему в списке узлу и т. д. За счет этого при отказе какого-либо узла в сети приложение продолжает работать без сбоев.

*«Схема островов».* Во второй схеме узлы взаимодействуют по принципу «каждый с каждым». На каждом из них есть своя популяция небольшого размера, популяции эволюционируют независимо друг от друга. Через указанный пользователем шаг алгоритма узлы посылают друг другу своих лучших индивидов. В такой схеме общее число итераций алгоритма уменьшается за счет расширения зоны поиска. Кроме того, снижается риск преждевременной сходимости к локальному минимуму за счет добавления новых генов в генотип популяции.

Кроме варианта, где острова развиваются по одинаковой схеме, у пользователя есть возможность выбрать схему островов с весами. В этом случае выделяется один узел, в котором не происходит мутация. На нем развивается популяция, накапливающая лучшие решения. На остальных островах расположены популяции, для которых мутация происходит с заданной пользователем частотой и количеством мутирующих за раз особей. Эти популяции нужны для исследования новых областей поиска, что повышает вероятность обновления генотипа популяции. Однако необходимо учитывать, что время от времени происходит дублирование накапливающей популяции на остальные узлы сети на тот случай, если главный узел выходит из строя, это требует некоторого времени при выполнении и памяти машин.

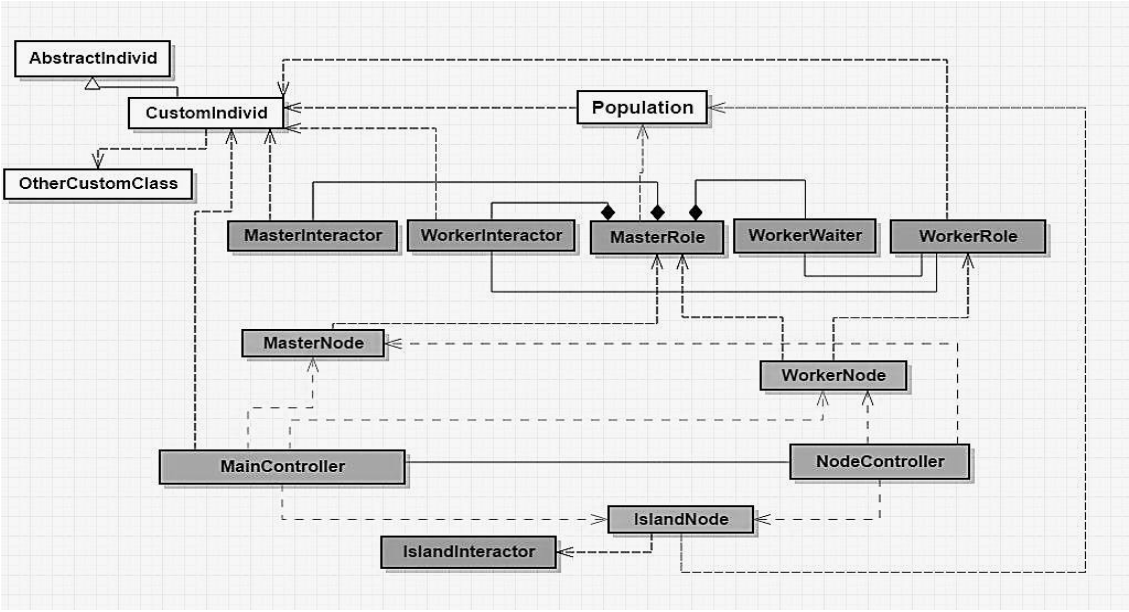
Описанные выше схемы естественным образом объединяются в «*Гибридную схему*». В ней несколько популяций развиваются независимо на своей группе узлов. В каждой группе узлов выделен свой мастер и свои рабочие. В случае отказа одного из мастеров первый его рабочий становится мастером и подключается к мастерам из других групп. Остальные рабочие в данной группе подключаются к новому мастеру.

## АРХИТЕКТУРА КЛАССОВ ПРИЛОЖЕНИЯ

Для реализации описанных выше схем была разработана архитектура классов, представленная на рисунке. Как уже было сказано выше, AbstractIndivid – предок создаваемого пользователем класса-особи. Пользователь формирует пакет, содержащий класс особи (на рисунке обозначен как CustomIndivid) и вспомогательные классы, и передает его главному модулю MainController. MainController взаимодействует с NodeController – модулем, который запускается на остальных компьютерах системы. Контроллеры сначала обмениваются данными для инициализации рабочего процесса, а затем запускают потоки MasterNode, IslandNode или WorkerNode, которые будут отвечать за организацию работы так называемых мастера, острова и рабочего соответственно. Выбор запускаемого потока зависит от указанных пользователем схемы и параметров алгоритма.

IslandNode производит эволюционный процесс и передает свои лучшие особи другим островам, используя потоки IslandInteractor, ответственные за взаимодействие с другими островами.

MasterNode запускает у себя поток MasterRole, который производит операции селекции, скрещивания и мутации. MasterRole содержит в себе поток WorkerWaiter, обрабатывающий подключения новых рабочих. Взаимодействие с каждым из рабочих осуществляется через объекты класса WorkerInteractor. Взаимодействие с другими мастерами в случае гибридной модели происходит с помощью потока MasterInteractor. WorkerNode запускает потоки WorkerRole и MasterRole. В этом случае MasterRole не производит операций ГА, а находится в состоянии ожидания рабочих. WorkerRole взаимодействует с мастером, посылая данные его потокам WorkerInteractor и WorkerWaiter.




Архитектура классов приложения

## АНАЛИЗ СИСТЕМЫ

В целях тестирования построенной системы были созданы два пакета с классами, необходимыми для описания особей в рамках двух различных задач. Одна из особей представляет собой формулу для аппроксимации функции по точкам, вторая – решение диофантова уравнения в виде массива значений переменных. Выбор таких особей для тестирования обусловлен возможностью контролирования сложности задачи, необходимого для выявления свойств системы. Так, в первом случае не сложно увеличить число точек, по которым будет аппроксимироваться некоторая функция, во втором – выбрать уравнение с большим количеством неизвестных.

В ходе тестирования приложения было выявлено, что при искусственном вызове отказа каких-либо узлов алгоритм продолжает работать и через некоторое время выдает корректный ответ, что свидетельствует о сохранности данных в системе. Для изучения скорости работы системы было запущено несколько тестов на разном количестве машин и для различной точности вычисления. В результате было выявлено, что при выборе распределенных алгоритмов наблюдается не только улучшение скорости

работы по сравнению с последовательным алгоритмом, но и уменьшение времени выполнения алгоритма с увеличением числа машин. Таким образом, наблюдается масштабируемость системы, т. е. есть возможность легко увеличить производительность за счет добавления новых узлов в сеть системы.

Можно прийти к выводу, что созданная система отвечает основным требованиям к распределенным системам: она реализует открытые интерфейсы, является прозрачной, т. е. представляется для пользователей как единая централизованная система, надежной и масштабируемой. Полученную систему можно использовать многократно для различных задач, решаемых генетическим программированием.

## ЗАКЛЮЧЕНИЕ

В данной статье была представлена созданная в ходе изучения распределенных вычислений система, позволяющая эффективно проводить действия ГА. Данная система является удобной для использования, надежной и дает возможность заметно увеличивать скорость вычислений за счет свойства масштабируемости. Ожидается, что построенное приложение будет использоваться при решении разнообразных задач неоднократно, сокращая время разработки реализаций ГА.

## БИБЛИОГРАФИЧЕСКИЕ ССЫЛКИ

1. Панченко Т. В. Генетические алгоритмы. Астрахань : Астрахан. ун-т, 2007.
2. Косяков М. С. Требования к распределенным системам // Введение в распределенные системы. СПб. : Питер, 2014. С. 13–17.